IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
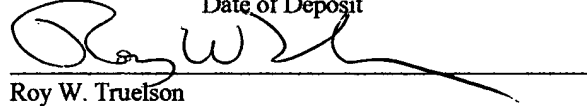
In re application of:	David B. Orchard, et al.	:	Date: April , 2005
Group Art Unit:	2122	:	IBM Corporation
Examiner:	J. Rutten	:	Intellectual Property Law
Serial No.:	09/838,620	:	Dept. 917, Bldg. 006-1
Filed:	April 19, 2001	:	3605 Highway 52 North
Title:	METHOD FOR DATA ACCESS CODE GENERATION	:	Rochester, MN 55901

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

DECLARATION OF INVENTOR (37 C.F.R. 1.131)

I, Arvind Viswanathan, do hereby declare as follows:

I am one of the named inventors in the above-entitled patent application, and am familiar with the circumstances of the creation of the present invention.

Docket No.: CA920000010US1
Serial No.: 09/838,620

-1-

After December 8, 1993, and prior to January 14, 2000, David B. Orchard and I conceived and reduced to practice the invention described and claimed in the above identified patent application, conception and reduction to practice of the invention occurring in Canada and/or the United States. Further evidence regarding conception and reduction to practice of the invention described and claimed herein follows.

At all times from the conception of the invention to the present, I have been employed by International Business Machines Corporation (IBM) of Armonk, N.Y., USA, in the capacity of a technical professional, assigned to IBM's facility in Vancouver, British Columbia, Canada. IBM requires its technical professionals to disclose to its management any inventions made in the course and scope of their employment which may have patentable value. For this purpose, IBM provides its employees with access to an electronic invention disclosure submission tool. On my information and belief, electronic submissions generated using this tool are records kept by IBM in its normal course of business.

Using this tool, I created and submitted a description of the concept of the invention claimed herein on or about October 25, 1999, which was assigned reference number CA 8-1999-0098. The attached document, entitled "Selective Portion of Database Record for Invention Disclosure CA 8-1999-0098", is a printed version of selective textual description of the invention which I submitted on or about October 25, 1999, evidencing conception of the invention. To the best of my information and belief, the text shown in the attached document was submitted by me on or about October 25, 1999; however, the printed version of the document does not necessarily preserve all formatting and graphics associated with the electronic database record, and certain additional information contained in the electronic database record has been deleted from the printed description.

At the time of submitting the electronic disclosure bearing reference number CA 8-1999-0098, i.e., October 25, 1999, the invention described and claimed herein had already been reduced to practice and embodied as an internal tool for use within IBM in generating computer

Docket No.:   CA920000010US1
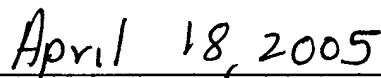Serial No.:   09/838,620

programming code. Computer programming code produced with the use of this internal tool (although not the internal tool itself) had already been provided externally to the University of Minnesota for use in a project called "StudentServer".

As further evidence of reduction to practice, referring to the final paragraph of the electronic disclosure bearing reference number CA 8-1999-0098, it is noted that the invention was: "Implemented as described in disclosure and used on IBM StudentServer project for Release 1 (Oct 1998)".

I hereby declare that all statements made herein of my own knowledge are true, and all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issuing thereon.
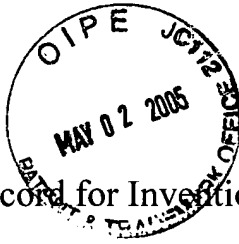
_____
Arvind Viswanathan

1

2

_____
(Date)

Docket No.:   CA920000010US1
Serial No.:   09/838,620

Selective Portion of Database Record for Invention Disclosure CA 8-1999-0098
Submitted October 25, 1999

Title of disclosure (in English)
Data Object Code Generation

1. Describe your invention, stating the problem solved (if appropriate), and indicating the advantages of using the invention.

Data Object Code Generation is the process whereby a data object can be described using XML, and processed during build time to create the necessary Java classes, which provide access to the backend data at run-time

2. How does the invention solve the problem or achieve an advantage,(a description of "the invention", including figures inline as appropriate)?

Data Object Code Generation

Overview

Data Object Code Generation is the process whereby a data object can be described using XML, and processed during build time to create the necessary Java classes, which provide access to the backend data at run-time. The advantages that this process has relative to the manual creation of the code include:

·· Simpler, straightforward set of XML elements can be used to describe the backend attributes and the mapping to Java objects.
·· More maintainable and consistent generation of code, which is especially useful when there are several data objects to manage
·· Faster generation of data objects
·· Allows optimization of the code in the future, for example, lazy initialization
·· Allows retargeting objects for a different backend
·· Users do not need to have knowledge of proprietary backend APIs
·· Allows generation of test suites

The overall Data Object Code Generation process is as follows:
1. Create an XML file describing the data object
2. Run the GenerateCode utility against the XML file created in the preceding step. The GenerateCode utility applies the code generation templates

Figure 1: Data Object Code Generation Process

Data Object Description in XML

Figure 1 shows the steps involved in using the code generator. First the author must use the data model that exists in the backend to create the data object description XML document. All data object description documents must conform to a single DTD which the Data Object Code Generator supports. The elements and attributes defined in the DTD depend on the backend access APIs. For example, a system that accesses a relational database backend will contain attributes and elements for the column names and the mapping of the columns to Java attributes.

A sample data object description XML for the *Career* object that retrieves data from the *ps_acad_car_vw* database table follows.

Figure 2: Career.xml - Sample data object description in XML

The *property* elements specify the attributes of the Java class. The *key* attribute of the *property* element defines whether the property is a key. If an attribute is a key, the value of the attribute cannot be altered after it the object is created. Key attributes are used to uniquely identify an object; the properties with *key=""true""* together form the primary key.

The *DatabaseAttributes* element contains the mapping of the Java attributes to the database column names. For each *input* and *output* tag within the *DatabaseAttributes* element, the author specifies the *property* and the *identifier* attributes. *property* contains the Java attribute name and *identifier* contains the backend identifier (a database column name in this example).

The *ReferenceAttributes* element describes an association relationship, which is implemented as a Java reference. For example, the *ReferenceAttributes* element is used to represent a foreign key relationship in a relational database. The elements within a *ReferenceAttributes* element contain the mapping of the attributes in the current object to the key (primary key or search key) attributes in the referenced object. *ReferenceAttributes* elements in the data object description are used to express a directed, labeled graph of objects, where each *ReferenceAttribute* element is a vector in the graph.

Data Object Data Type Definition (DTD):

Figure 3: DataObject.dtd - DTD for data object description in XML

Figure 3 contains the DTD for the data object description shown in the Career.xml. In addition to *DatabaseAttributes* for accessing database attributes, the DTD also allows *PeopleSoftAttributes* for describing the access to PeopleSoft panels. *PeopleSoftAttributes* can contain an additional element called *subPanel*, which is used to retrieve the attributes in a nested panel.

The *Career* object in the previous example can be retargeted to serialize information to/from PeopleSoft instead of the database by simply modifying the Career.xml data object description file. The *DatabaseAttributes* element will be replaced with a *PeopleSoftAttributes* element. Once Career.xml has been updated with the *PeopleSoftAttributes* element, rerunning the code generator routine will produce code that uses the PeopleSoft API to serialize the object instead of using JDBC to serialize to the database.

Code Generation Templates

The code generation templates shown in Figure 1 describe the code that is generated when the code generation routine is invoked. The code generation routine takes the code generation template and the data object description XML as inputs and produces Java source code. The code generation template contains the following:

·· API calls for the backends that are supported in the DataObject.dtd. For the example in Figure 2, the code generation template will contain support for PeopleSoft Message Agent API calls for accessing PeopleSoft and JDBC API calls for accessing relational databases.
·· Implementation of system-wide policies such as caching, lazy initialization, logging and use of system infrastructure.
·· Rules for code generation. For example, if a property is defined as a key, then only the getter method for the attributes is generated.

Describing a sub-graph of data objects for improved performance

The *ReferenceAttribute* elements can be used to describe a directed, labeled graph. Each data object description XML document is a node and the *ReferenceAttribute* elements are vectors of the graph. Each generated data object class typically performs lazy initialization. This approach is more efficient than populating the entire graph of objects, but it can result in several backend accesses.

Figure 4: Defining a sub-graph of data objects

For the object model in Figure 4, assume that data objects A, B and C need to be retrieved. The number of queries to retrieve data objects A, Band C using lazy initialization is as follows:

	Number of queries to retrieve objects	Number of objects returned
A	1 (find by primary key)	A
B	a	B
C	ab	C

In general,

Number of backend accesses to retrieve object hierarchy $(A\text{->}B\text{->}C......) = 1 + a + ab + abc.......$

where a, b, and c are the number of objects of class A, B, and C, respectively.

Therefore, the number of queries to retrieve an object hierarchy is proportional to the number of objects when lazy initialization is used.

In situations where there is considerable overhead for each access to the backend and the backend is efficient at performing joins, the overall systems performance can be improved by performing a single query that retrieves all the objects in the sub-graph of interest.

Figure 5. Sample sub-graph description in XML

The sub-graph of the data model is described in an XML document, like the one shown in Figure 5. The example describes a sub-graph for the *CurrentEnrollmentsView* business function that will retrieve data objects A, B and C using a single query. The example assumes that *ReferenceAttribute* elements from A to B and from B to C exist in the data object description XML documents for A and B respectively. Notice that data object D is not retrieved as part of this sub-graph.

The DataObjectTree element identifies the business function that this sub-graph or view of the model is for. Each *rootnode* element identifies a starting point of traversal of the object model. The *rootnode* elements contain *reference* elements to indicate that automatic pre-fetching of that reference is required. For example, *<reference name=""dataObjectB"" dataObject=""B"">*, specifies that the reference from data object A to B should be retrieved as part of the single query. *reference* elements may be nested.

Using a code generation template that handles *DataObjectTree* documents, Java code can be generated to perform a single, large query that retrieves data objects A, B and C. The template will use the *ReferenceAttribute* element from A.xml, B.xml and C.xml to generate the correct join attributes for the single query.

The description of the data object in XML makes it possible to perform enhancements such as the use of sub-graphs for backend access optimization.

Conclusion

Using XML to describe data objects has several advantages. The users of the Data Object Code Generator need not learn the proprietary backend APIs, since the proprietary calls are contained only within the code generation template. The users of the code generator are then able to focus on leveraging their knowledge of the backend data model to create data object objects more efficiently. System-wide changes and enhancements can be made as part of a new build by regenerating the data objects using a modified code generation template or data object description. Using XML to describe the data object allows validation against the data object DTD to catch errors early in the development.

. . .

4. If the invention is implemented in a product or prototype, include technical details, purpose, disclosure details to others and the date of that implementation.

Implemented as described in disclosure and used on IBM StudentServer project for Release 1 (Oct 1998)